

Learning Sequence Neighbourhood Metrics

Justin Bayer, bayer.justin@gmail.com
 Christian Osendorfer, osendorf@in.tum.de
 Patrick van der Smagt, smagt@tum.de

September 12, 2011

Abstract

Recurrent neural networks (RNNs) in combination with a pooling operator and the neighbourhood components analysis (NCA) objective function are able to detect the characterizing dynamics of sequences and embed them into a fixed-length vector space of arbitrary dimensionality. Subsequently, the resulting features are meaningful and can be used for visualization or nearest neighbour classification in linear time. This kind of metric learning for sequential data enables the use of algorithms tailored towards fixed length vector spaces such as \mathbb{R}^n .

1. Introduction

Sequential data is found in many domains including medical applications, robot control, neuroscience, financial information or text processing. This data is fundamentally different from static data vectors.

When considering a single sequence over T time steps $\mathbf{x} = (x_1, x_2, \dots, x_T) \in X^*$ with $X \subset \mathbb{R}^n$, the order of the individual elements x_i is relevant for the interpretation. Conversely, in the case of static data $\mathbf{x}' \in \mathbb{R}^n$, an ordering on the n components is not even defined. Indeed, the key element of structured data is that the context (i.e., dependency between the time steps) contains essential information to make learning on the data possible.

An often used simplifying approach is to treat the data as if it were static. As we will demonstrate, this assumption makes the detection of behaviour typical for sequences impossible. A more promising and principled way is to extract higher-order features from the sequences. For instance, given a sequence of robot joint positions $q(t)$, standard methods can be used to calculate derivatives $q'(t)$ and $q''(t)$, which can then be used as additional data. Alternatively, domain knowledge of experts can be used. Nonetheless, such methods usually require an innate knowledge of the underlying, data-generating process, which is not

always available or efficient. Turning to algorithmically learned features is a logical step.

The authors of (Goldberger et al. 2004) note that metrics and features are actually closely related: by measuring pairwise distances between the data points $\mathbf{x}' \in \mathbb{R}^n$, the data can be embedded into a metric space. They learn a Mahalanobis distance by mapping the high-dimensional data set X to a metric space Z in which k -nearest neighbour classification performance is maximized. The resulting objective function is differentiable with respect to the embedding.

Similar to (Salakhutdinov and Hinton 2007), we use a different model for learning the embedding function. Our choice, recurrent neural networks (RNNs), are rich models for sequence learning. They have been successfully used for handwriting recognition (Graves and Schmidhuber 2009), audio processing (Graves and Schmidhuber 2005), and text modelling (Martens, Sutskever, and Hinton 2011). Although in principle capable of approximating any measurable sequence-to-sequence mapping (Hammer 2000), they are notoriously hard to train. We successfully apply them to several data sets by making use of the most recent version of a special architecture, the Long Short-Term Memory (Hochreiter and Schmidhuber 1997), which overcomes the learning difficulties.

2. Characteristics of Sequential Data

We consider a *sequence* $\mathbf{x}_t \in X \subset \mathbb{R}^n$, with $t = 1, 2, \dots, T$, where X is called the *sequence space*. The sequence space can be a representation of time series, as well as nominal data such as text (using “1-of- k ” encodings).

2.1. Requirements for Sequence Metrics

Ideally, a metric would reflect the different axes of the underlying process and only those. To illustrate the difficulty of this, we give a small number of examples of what nature these dynamics might be and what a learning machine has to be able to detect. The concepts are visualized in figure 1.

1. Time lags are essentially *translations* along the time axis. These translations might occur in the middle of a sequence, not only at the beginning or at the end. Distances that are a sum of the pairwise distances of the form $D(\mathbf{x}, \mathbf{y}) = \sum_i d(x_i, y_i)$ such as the Euclidean (with $d(x, y) = \sqrt{(x - y)^2}$) or the Hamming distance (with $d(x, y) = 1 - \mathbb{I}(x = y)$) are unable to capture this. Sequences might as well be translated in sequence space as a whole. The essential part of a sequence might be that it increases over the whole time span and thus depends on the difference of two succeeding values and not on their respective differences from some arbitrary origin.

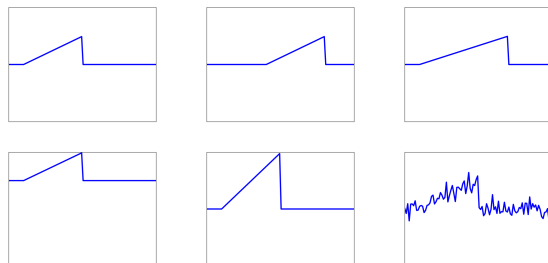


Figure 1: Examples of sequence characteristics. Top row, from left to right: basic sequence, translation along the time axis, scaling along the time axis. Bottom row: translation along the time axis, scaling along the time axis and additional Gaussian noise.

2. The axes might have different *scalings*. While the actual magnitudes of the values at the individual time steps are less significant, the topology induced by those values can be crucial. E.g., it might be necessary to detect how many spikes are in a sequence whilst the individual heights of those hills is of less interest for the task. More extreme cases include a drift over time, in which the axes might even be warped nonlinearly.
3. Lots of time series data are subject to *stochasticity*. While each time step might be distorted by a Gaussian noise term, noise can also occur along the time axis. E.g., the time span until a special event happens might be random itself.

A more general way of putting this central requirement is that the metric should be able to capture the underlying dynamics. The nature of these dynamics is more complicated than the previously mentioned points given in a lot of phenomena: Each of them can either be of negligible or of central importance for the given task.

2.2. Related Work

Only a few principled approaches exist for extracting fixed length features from sequential data. A commonly used practice is based on a set of fixed basis functions (e.g. Fourier or wavelet basis).

While it has strong mathematical guarantees, it is sometimes too inflexible: in order to work with arbitrarily long sequences, a sliding time window has to be employed, limiting the capability to model context.

Furthermore, the fixed set of basis functions implies that the problem of identifying useful factors of variations remains unresolved in general.

A probabilistic yet supervised approach, is to use a generative model. For each class c_i , a conditional $p(x|c_i)$ is estimated. The resulting models can be chosen to be capable of capturing the characteristics of the data at hand (e.g. hidden Markov models for speech data) and are combined with class distributions $p(c_i)$ into a Bayesian classifier. The predictive distribution is then given by $p(c_i|x_i) = \frac{p(x_i|c_i)p(c_i)}{p(x_i)}$. The resulting posterior likelihoods can then be interpreted as features. Essentially class memberships, they can be deemed not expressive enough as they are very abstract.

Fisher kernels (Jaakkola and Haussler 1998), a combination of probabilistic generative models with kernel methods, provide another commonly vectorial representation of sequences. The basic idea is that two similar objects induce similar gradients of the likelihood for the parameters of the model.

Thus, the features for a sequence are the elements of the gradient of the log-likelihood of this sequence with respect to the model parameters.

This choice can presumably be very bad: if the distribution represented by the trained model closely resembles the data distribution the gradients for all sequences in the data set will be nearly zero. A recent paper (Maaten 2011) alleviates this problem by exploiting label information and employing ideas from metric learning. Obviously, this only works if class information is available.

A fully unsupervised approach is to use the parameters estimated by a system identification method (e.g., a linear dynamical system) as features. Recent work includes (Li and Prakash 2011), in which a complex numbers based system successfully clusters motion capture data.

The last two approaches clearly suffer from the fact that the number of features is directly connected with the complexity of the model. In particular it is not given that the important factors of variation are captured by these methods.

3. Recurrent Neural Networks

Recurrent neural networks are an extension of feedforward networks which can represent sequential data by having an internal state. While a state-free feedforward network immediately “forgets” the data it has seen, RNNs have weighted connections through time, which means that information can be kept over the forward propagation of a single input vector. The inputs to an RNN are given as a sequence (x_1, x_2, \dots, x_T) . Subsequently, a sequence of hidden states (h_1, h_2, \dots, h_T) and a sequence of outputs (o_1, o_2, \dots, o_T) is calculated via the following equations:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$

$$o_t = W_{ho}h_t + b_o \quad (2)$$

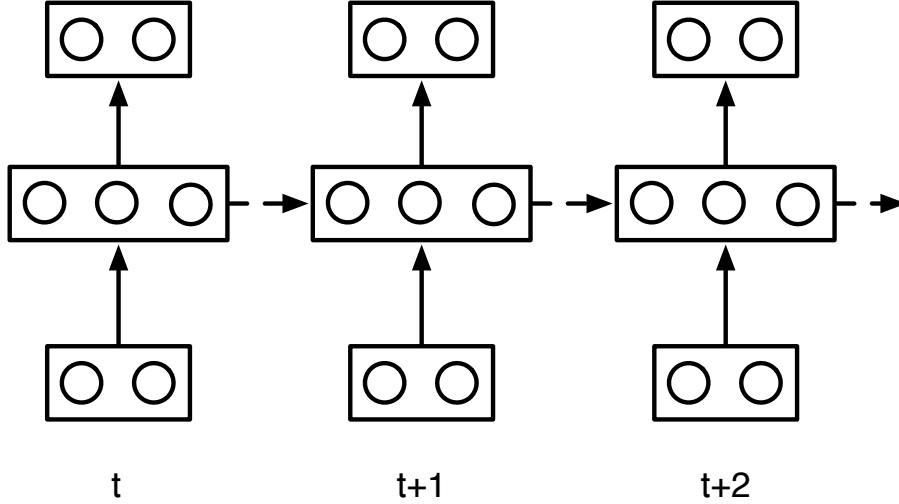


Figure 2: Left: Structure of an ordinary RNN. Dashed lines represent recurrent connections. Right: LSTM-RNN with a single cell. The red dots represent the states s_t , while to arrows represent summing connections. The dotted connections represent multiplicative interactions. Peepholes left out for clarity.

where $t = 1, 2, \dots, T$ and σ is a suitable transfer function, typically the tangent hyperbolic, applied element-wise. W_{xh} , W_{hh} , W_{ho} are weight matrices while b_h and b_o are bias terms. For the calculation of h_1 a special initial hidden state h_0 has to be used which can be optimized during learning as well.

The dimensionality of the adaptable parameters W_{xh} , W_{hh} , W_{ho} , b_h , b_o and h_0 is determined by the given input and output dimensions and the size of the hidden layer. Given an input size I , a hidden size H and an output size O the following dimensionalities are met: $W_{xh} \in \mathbb{R}^{I \times H}$, $W_{hh} \in \mathbb{R}^{H \times H}$, $W_{ho} \in \mathbb{R}^{H \times O}$, $b_h, h_0 \in \mathbb{R}^H$ and $b_o \in \mathbb{R}^O$.

The structure of RNNs is illustrated in figure 2.

RNNs have a lot of expressive power since their states are distributed and nonlinear dynamics can be modelled. The calculation of their gradients is astonishingly easy via Backpropagation Through Time (BPTT) (Mozier 1989) or Real-Time Recurrent Learning (Williams and Zipser 1995). The guiding mathematical tool is the chain rule, which can be applied “through time” as well. However, 1st order gradient methods completely fail to capture relations that are more than as little as ten time steps apart of each other.

This problem is called the *vanishing gradient* and has been studied by (Hochreiter 1991) and (Bengio, Simard, and Frasconi 1994). The state of the art method to overcome this has been the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) for more than ten years. Recently (Martens and

Sutskever 2011) introduced a second-order optimization method for RNNs, the Hessian free optimizer (HF-RNN), which is able to cope with aforementioned long term dependencies as well, outperforming LSTM on several benchmarks. In this work, we stick to LSTM since the HF-RNN is tailored towards convex loss functions—neighbourhood component analysis (NCA), the objective function of choice in this paper, is however not convex.

Another neural model for nonlinear dynamical systems is the echo state network approach introduced in (Jaeger and Haas 2004). The drawback of this method is that the dynamics that are to be modelled have to be already present in the network’s random initialization.

3.1 Recurrent Networks are Differentiable Sequence Approximators

One consequence of the differentiability of RNNs is that we can optimize their parameters with respect to an objective function.¹ Stochastic gradient descent or higher order techniques are the techniques of choice to fit the weights.

A long overlooked but obvious potential is to reduce output sequences to a single vector with a *pooling operation*. A pooling operation is a function $p : X^* \rightarrow X$ that reduces an undefined amount of inputs to a single output of the same set, e.g. taking the sum or picking the maximum. Similar to convolutional neural networks, we can use this technique to reduce a sequence to a point. If our pooling operation is differentiable as well, we can use it as a gateway to arbitrary objective functions that are defined on real vectors. Given a network f parametrized by W , a data set $\mathcal{D} = \{x_i\}$, a pooling operation p and an objective function \mathcal{O} we proceed as follows:

1. Process input sequences $\mathbf{x}_i = (x_{i1}, \dots, x_{iT}), x_{it} \in \mathbb{R}^n$ to produce output sequences $f(\mathbf{x}_i; W) = \mathbf{o}_i = (o_{i1}, \dots, o_{iT}), o_{it} \in \mathbb{R}^m$,
2. Use a pooling operation p to reduce the output sequences to a points via $p(o_{i1}, \dots, o_{iT}) = e_i$,
3. Calculate the objective function $\mathcal{O}(\{e_i\})$.

Since the whole calculation is differentiable, we can evaluate the derivative of the objective function with respect to the parameters of the RNN via

$$\frac{\partial \mathcal{O}}{\partial p} \frac{\partial p}{\partial f} \frac{\partial f}{\partial W}. \quad (3)$$

Subsequently, we can use the gradients to find embeddings $\{e_i\}$ of our data which optimize the objective function. We apply this insight to combine RNNs

¹The authors recommend to use automatic or symbolic differentiation. In this work, Theano (Bergstra et al. 2010) was used.

with neighbourhood components analysis (NCA), which we will introduce in section 4.

3.2 Long Short-Term Memory

The capability of LSTM cells to relate events in sequences more than hundreds of time steps apart is attributed to a special building block. These so called *gating units* implement a differentiable version of the `if ... then ...` construct found in programming languages. We define $\phi(c, v) = v\sigma(c)$ with σ being the sigmoid function $\frac{1}{1+e^{-x}}$ ranging from 0 to 1. Here, c can be seen as the *condition* that controls v : if c is very low (representing *false*) the output is 0. If it is very high (representing *true*) the output is v .

A central concept are the *states* (s_1, s_2, \dots, s_T) of the cell. These can be altered by the inputs via the *input*, *forget* and *output gate*. We will now give the formulas for a recurrent neural network with LSTM cells. To keep the notation uncluttered, we concatenated the four different inputs $a_t^{(\cdot)}$ to the cell into a single vector. As indicated by the superscript, each of the $a_t^{(\cdot)}$ represents an input to one of the gates i , f and o . The superscript x represents the input to the cell itself.

$$[a_t^{(x)} a_t^{(i)} a_t^{(f)} a_t^{(g)}] = W_{ha}h_{t-1} + W_{xa}x_t + b_a \quad (4)$$

$$s_t = \underbrace{\phi(a_t^{(i)}, a_t^{(x)})}_{inputgate} + \underbrace{\phi(a_t^{(f)}, s_{t-1})}_{forgetgate} \quad (5)$$

$$h_t = \sigma(\underbrace{\phi(a_t^{(o)}, s_t)}_{outputgate}) \quad (6)$$

$$o_t = W_{ho}h_t + b_h$$

Given an input size I , a hidden size H and an output size of O the parameters have the following dimensionalities: $W_{xa} \in \mathbb{R}^{I \times 4H}$, $a_t \in \mathbb{R}^h$, $s_t \in \mathbb{R}^h$, $W_{ha} \in \mathbb{R}^{H \times 4H}$, $b_a \in \mathbb{R}^{4H}$ and $b_o \in \mathbb{R}^o$. Recurrency is achieved twofold: first, in equation (4) via the weight matrix W_{ha} and second in the forget gate in the second term of equation (5). The latter connection is not parametrized and thus sometimes referred to as a *constant error carousel*.

A major improvement of the LSTM cells was the introduction of *peepholes* by (Gers, Schraudolph, and Schmidhuber 2003). Additional connections from the states to the gates make it possible to learn precise timings. This results in additional learnable parameters $p_i, p_f, p_o \in \mathbb{R}^H$. Letting \odot represent the pairwise multiplication of vectors, we change equations (5) and (6) in the following way:

$$s_t = \phi(a_t^{(i)} + p_i \odot s_{t-1}, a_t^{(x)}) + \phi(a_t^{(f)} + p_f \odot s_{t-1}, s_{t-1}) \quad (7)$$

$$h_t = \sigma(\phi(a_t^{(o)} + p_o \odot s_t, s_t)). \quad (8)$$

A network with LSTM cells is shown in figure 2.

4. Sequential Neighbourhood Components Analysis

The central assumption of neighbourhood components analysis (Goldberger et al. 2004; Salakhutdinov and Hinton 2007) is that items of the same class lie near each other on a lower-dimensional manifold. To exploit this, we want to learn a function $f : X \rightarrow Z$ from the sequence space X to a metric space Z that reflects this.

The resulting embeddings are tailored towards good performance in combination with the k -nearest neighbour algorithm. The embeddings are however not limited to this. Other approaches, such as DrLim (Hadsell, Chopra, and Lecun 2006) use the same idea in combination with energy based models to learn metrics. The resulting embeddings can be used in conjunction with any algorithm working on static data. Practical results have been shown that large margins separate the distinct classes. In some cases, points of the same class form multiple clusters.

In our case, the embedding function is given as $e(\mathbf{x}; W) = p \circ f$. A recurrent neural network f is used to map sequences over \mathbb{R}^I to sequences over \mathbb{R}^O of equal length. The resulting output sequence is then reduced to a single point via the pooling operation p .

Given a set of sequences with an associated class label $\mathcal{D} = \{x_i, c_i\}$ mapped to a set of embeddings $\mathcal{E} = \{e(x_i; W) = e_i\} \subset Z$, we define the probability that a point a selects another point b as its neighbour based on pairwise distances d_{ab} as

$$p_{ab} = \frac{\exp(-d_{ab})}{\sum_{z \neq a} \exp(-d_{az})}, \quad (9)$$

while the probability that a point selects itself as a neighbour is set to zero: $p_{aa} = 0$. The pairwise distances are determined by the Euclidean distances of the respective embeddings: $d_{ab} = \|e_a - e_b\|^2$.

The probability that a point i belongs to a certain class k depends on the classes of the points in its neighbourhood

$$p(c_i = k) = \sum_j p_{ij} \mathbb{I}(c_j = k),$$

where \mathbb{I} is the indicator function that returns 1 if the argument is true and 0 otherwise. We are now set to state the overall objective function: we maximize

the expected number of correctly classified points

$$\mathcal{O} = \sum_i \sum_j p_{ij} \mathbb{I}(c_i = c_j).$$

This objective can be optimized as shown in section 3.1 with equation (3).

It should be noted, that NCA is capable of adjusting the number of neighbours for each point: equation (9) shows that the contribution of a neighbour goes to zero exponentially fast with its distance. Thus, a simple scaling of the whole coordinate system of the embedding space can be perceived as a soft selection for the amount of neighbours to consider for a class prediction.

4.1 Classifying Sequences

We first train an RNN on our data set with the NCA objective function. Afterwards, all training sequences are propagated through the network and the pooling operator to obtain embeddings $\mathcal{E} = \{e_i\}$ for each of them. We then build a nearest neighbour classifier for which we use all embeddings of the training set. A new sequence (x_1, x_2, \dots, x_T) is classified by first forward propagating it through the RNN and obtaining an embedding. We then find the k -nearest neighbours and obtain the class by a majority vote.

The complexity of classifying a new sequence given a trained RNN is thus $\mathcal{O}(T) + C(f)$. $C(f)$ is the complexity of a nearest neighbour lookup working in an f -dimensional space. In contrast, the state of the art method for time series classification, dynamic time warping, has a complexity of $\mathcal{O}(T^2 N)$ where N is the number of sequences in the training data set.

5. Experiments

To show that our algorithm works as a classifier we present results on several data sets from the UCR Time Series archive (Keogh 2006). By analyzing the Cylinder Bell Funnel data set in more detail, we show that the extracted features are actually meaningful. Then we give results on more data sets from the UCR archive—although we do not necessarily reach top performance in terms of classification error on them, we can see that our algorithm does something reasonable. Outstanding results on all of the UCR data sets are achieved by Dynamic Time Warping (DTW) in combination with nearest neighbour classification (for an overview see (Xi et al. 2006)).

We then proceed to a real-world data set, namely TIDIGTS to show that our method is useful for practical applications. TIDIGTS is similar to the widely used data set MNIST as it contains the digits 0 to 9, yet spoken instead of

written. For TIDIGITS, the performance of a discriminative model based on LSTM-RNNs has been reported in (Graves and Schmidhuber 2005).

We optimized the parameters with resilient backprop (RPROP) (Riedmiller 1994). In our experiments, ordinary stochastic gradient descent with momentum was never able to find a good parameter set. RPROP on the other hand found good minima after initial periods where the training error did not improve significantly. It also proved very robust towards hyperparameter selection. In all experiments, a maximum step size of 1.0, a minimum step size of 10^{-6} , a growth factor of 1.1 and a shrinking factor of 0.3 was used. All input data was normalized to have zero mean and unit variance. The dimensionality of the embeddings was 2 if not mentioned otherwise. As a pooling operator, we chose the average: $p(o_1, \dots, o_T) = \frac{1}{T} \sum_i o_i$.

5.1 Synthetic Data: UCR Time Series

As noted in (Goldberger et al. 2004) we confirmed that NCA almost never overfits in our experiments. Thus, all experiments were conducted by training several runs until convergence on the training set. We then report the testing error of the set of parameters that performed best on the training set.

The cylinder bell funnel data set is a synthetic data set that includes all the problems with sequences outlined in section 2: translation, scaling and stochasticity. It consists of equal length, one dimensional time series of three classes. At a random position a motif (either cylinder, bell or funnel) occurs for a random period of time.

In the case of a *cylinder*, the value of the time series increases suddenly and stays on its level for a while after which it decreases suddenly to the previous level.

A *bell* is characterized by a slow, constant increase over several time steps and a sudden drop to the base level again. *Funnels* are quite the opposite of a bell: abrupt increasement and slow decreasement to the baselevel. All items in the data set are also subject to Gaussian noise. For the exact formulas, see (Geurts 2002).

Although the training data is very limited, consisting of only ten sequences per class, we can see that the classes are separated in a meaningful way: The vertical axis corresponds to the length of the cylinder, bell or funnel. The horizontal axis represents the transition from bell over cylinder to funnel. This is illustrated in figure (3).

We now give the classification results and specific parameters for several more data sets.

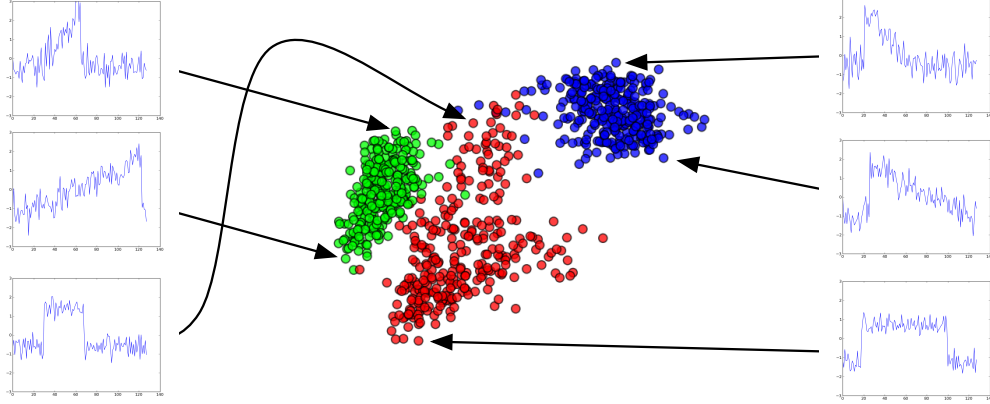


Figure 3: Embeddings found for test sequences of the CBF data set. The green cluster represents the bell class, the red cluster the cylinder class and the blue cluster the funnel class. Individual sequences have been pointed out to show that the vertical axis roughly corresponds to the length of the object while the horizontal is indicative of the transition from bell over cylinder to funnel time series.

Data set	# classes	# LSTM cells	Epochs	Training error	Testing error	1KNN
CBF	3	12	97	0.972	0.971	0.957
Lighting 2	2	10	100	0.964	0.588	0.721
Lighting 7	7	10	55	0.534	0.458	0.493
Two Patterns	4	5	102	0.676	0.656	0.694

The training and test errors stated are the average probabilities that a point is correctly classified by the stochastic classifier used in the formulation of NCA. We also report the error for 1-nearest neighbour classification.

5.2 Real World Data: TIDIGITS

TIDIGITS is a data set consisting of spoken digits by adult and child speakers. We restricted ourselves to the adult speakers. The audio was preprocessed with mel-frequency cepstrum coefficient analysis. The setup parameters were 12 cepstral coefficients, 1 energy coefficient, and 13 first derivatives, giving 26 coefficients in total. The frame size was 15 ms and the input window was 25 ms. We used a framesize of 15 ms and an input window of 25 ms.

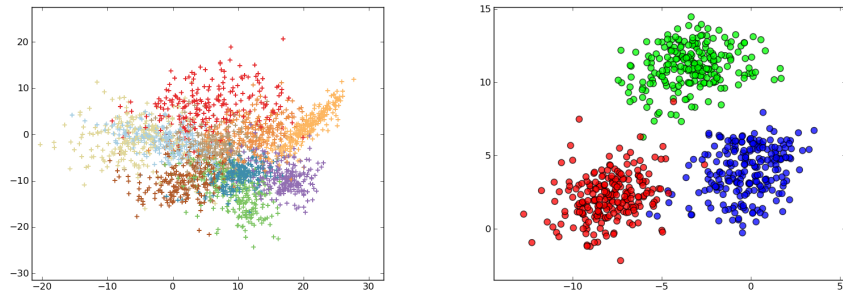


Figure 4: Two dimensional embeddings for the TIDIGITs data set. Left: the whole data set with all ten classes. Right: Only the classes corresponding to the digits "1", "2" and "4".

We trained our network on the full data set and on a subset containing only the digits "1", "2" and "4". In both cases, we trained networks with 40 LSTM units for 250 epochs.

Data set	Training error	Testing error	1KNN
3 digits	0.979	0.981	0.984
All digits	0.601	0.522	0.584

Although these errors are not state of the art, we want to point out that discriminative models based on LSTM-RNNs are in fact able to get correct classification rates of more than 99% (Graves and Schmidhuber 2005). As can be seen in Figure (4), RNNs are indeed able in conjunction with NCA to find low level representations on real world sequential data.

6. Conclusion

We presented a solution to an important problem—by combining two well established methods we introduced a method to embed sequential data into a semantically meaningful metric feature space. Despite not achieving state of the art performance on widely used benchmark data sets our method has its own value: it has significantly lower complexity than DTW, the current state of the art. Furthermore, it is parametric and thus much easier to use for big or non-stationary data. Additionally it leads to interpretable features naturally and can be used out of the box as a visualization method and data exploration tool.

The techniques presented here are usable with any RNN structure—we believe that the usage of echo state networks (Jaeger and Haas 2004) or multiplicative RNNs (Martens, Sutskever, and Hinton 2011) to NCA might yield even better results. The visualizations and performances in (Salakhutdinov and Hinton 2007) are arguably more impressive. But it should be noted that a recurrent pendant to deep belief networks is an unaddressed problem—our work would definitely benefit from its solution.

References

- Bengio, Y., P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5: 157–66. doi:10.1109/72.279181.
- Bergstra, James, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU Math Expression Compiler. *Proceedings of the Python for Scientific Computing Conference (SciPy)*. http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf.
- Gers, Felix A., Nicol N. Schraudolph, and Jürgen Schmidhuber. 2003. Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research*.
- Geurts, P. 2002. Contributions to decision tree induction: bias/variance tradeoff and time series classification.
- Goldberger, Jacob, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. 2004. Neighbourhood components analysis. *Advances in Neural Information Processing Systems 17*. MIT Press.
- Graves, Alex, and Jürgen Schmidhuber. 2009. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. *Neural Information Processing Systems*: 545–52.
- Graves, Alex, and Juergen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18: 602–10. doi:10.1016/j.neunet.2005.06.042.
- Hadsell, Raia, Sumit Chopra, and Yann Lecun. 2006. Dimensionality Reduction by Learning an Invariant Mapping. *Computer Vision and Pattern Recognition*. doi:10.1109/CVPR.2006.100.
- Hammer, Barbara. 2000. On the Approximation Capability of Recurrent Neural Networks. *ICSC Symposium on Neural Computation*.
- Hochreiter, S. 1991. Untersuchungen zu dynamischen neuronalen Netzen.
- Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9: 1735–80. doi:10.1162/neco.1997.9.8.1735.
- Jaakkola, Tommi, and David Haussler. 1998. Exploiting Generative Models in Discriminative Classifiers. In *Advances in Neural Information Processing Systems 11*. MIT Press.

- Jaeger, Herbert, and Harald Haas. 2004. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science* 304: 78–80. doi:10.1126/science.1091277.
- Keogh, Xi, X. ., Wei, L. & Ratanamahatana, C. A., E. 2006. The UCR Time Series Classification/Clustering Homepage. http://www.cs.ucr.edu/~eamonn/time_series_data/.
- Li, Lei, and B. Aditya Prakash. 2011. Time Series Clustering: Complex is Simpler!.
- Maaten, Laurens van der. 2011. Learning Discriminative Fisher Kernels. *Proceedings of the 28th International Conference on Machine Learning*.
- Martens, James, Ilya Sutskever, and Geoffrey Hinton. 2011. Generating Text with Recurrent Neural Networks. *Proceedings of the 28th International Conference on Machine Learning*.
- Martens, James, and Ilya Sutskever. 2011. Learning Recurrent Neural Networks with Hessian-Free Optimization. *Proceedings of the 28th International Conference on Machine Learning*.
- Mozer, M. C. 1989. A Focused Backpropagation Algorithm for Temporal Pattern Recognition.
- Riedmiller, Martin. 1994. Rprop - Description and Implementation Details.
- Salakhutdinov, Ruslan, and Geoffrey Hinton. 2007. Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure.
- Williams, Ronald J., and David Zipser. 1995. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity.
- Xi, Xiaopeng, Eamonn J. Keogh, Christian R. Shelton, Li Wei, and Chotirat Ann Ratanamahatana. 2006. Fast time series classification using numerosity reduction. *International Conference on Machine Learning*. doi:10.1145/1143844.1143974.